New Algorithm: Model-based RL with DDPG for Arbitrarily Positional Reaching

Lu Chen*, Yang Chen*, Yating Lin*, Ruiyi Wang*, Xinyu Wang*, Zhuoran Yang*, Ying Yuan*

Abstract

Most existing pioneered model-based reinforcement learning (RL) algorithms often achieve high efficiency by providing predictive models. Such algorithms allow the allocation of computational resources to be reduced, however, they are also limited to high-dimensional continuous space and sparsely sampled data. Moreover, if we directly design a model-based RL algorithm, the ability to extend to high-capacity models is significantly limited. In this project, we propose and implement a new reinforcement learning algorithm for arbitrarily positional reaching, called Model-based MPC RL with DDPG. We demonstrate improvements of the new algorithm in training efficiency both in sparse and dense reward conditions.Furthermore, based on our current algorithm, we also proposed a new advanced version of the model-based RL algorithm, called Model-based MPPI RL with DDPG and HER, showing the potential possibility to apply the model-based RL with the model-free algorithm in the real-time and real-world learning.

1 Introduction

Recent years have witnessed rapid development on robotic tasks due to the popularity of varying algorithms. Numerous robotic tasks can be framed as RL problems, where a robot aims to select an optimal policy and optimize a cost function. To solve these problems, the methods are often categorized into model-free and model-based approaches.

Model-based RL algorithms can learn a wide range of tasks. Such algorithms provide high-efficiency learning, by learning a model of the system and training a feedback control policy based on the learned model. Model predictive control (MPC) is one of the most effective ways to achieve generalization for model-based RL tasks, relying on online optimization of the cost function. A more flexible MPC method is model predictive path integral (MPPI) control. However, most variations of MPC rely on tools from constrained optimization, meanwhile, it is hard to extend model-based algorithms to high-capacity models. In comparison, model-free algorithms are able to perform well on a wide range of tasks. Model-free approaches to RL, such as Deep Deterministic Policy Gradient (DDPG) and Twin Delayed DDPG (TD3) methods, have been successfully applied to many challenging tasks. However, such good performances require a large number of samples.

We consider the main reason for this issue are the challenges caused by high-dimensional manipulation tasks, where the environment has high-dimensional continuous space with limited samples and expensive experiences. In this project, we provide a new algorithm for this issue. The intuitive idea is to make the robot use virtual experiences generated by some models where the robot is trained to reach objects using arms. In this way, the manipulation task is performed by combining both model-based RL and model-free RL algorithms. More specifically, we first use real experience to train the dynamic model. Then, we utilize the trained model to generate virtual experience. After getting virtual experience and real experience, we perform model-free algorithms to update policy.

*All authors have equal contributions.

2 Related Work

Model-free algorithms have proved to perform well on a wide range of robotic tasks, but such performance is dependent on a great number of samples. In the field of Robotics, the high-dimensional continuous space and limited samples and experiences hinder researchers from implementing large-scale training of model-free algorithms to fulfill the desired purposes. In comparison, model-based algorithms are capable of learning efficiently based on a smaller scale of available data and less complex configuration of the interactions with the environment. Many lines of research attempted to analyze the behavior of model-based reinforcement learning. Kaiser et al. (2020) designed a model-based deep RL algorithm based on video prediction models, where the agent learns from imaginary experiences generated from the predictive model. Zou et al. (2020) integrated model-based reinforcement learning and direct offline learning to mitigate expensive computing and instability on convergence. The attempts to combine model-based algorithms with deep neural network revealed importance of maintaining stability in training and constructing high-quality predictive models.

Deep Deterministic Policy Gradient (DDPG) is a model-free, off-policy actor-critic algorithm used in a continuous control environment. DDPG improves the vanilla actor-critic technique by utilizing the Deterministic Policy Gradient (DPG) in the Deep Q Network (DQN) (Lillicrap et al., 2016). In contrast with DQN that learns indirectly through Q-tables, DDPG learns directly from the off-policy data through policy gradient methods which better tackles the continuous action space problem. In order to meet the challenge of unstable learning and fragile hyperparameter tuning, a set of improvements over DDPG are invented. Twin Delayed DDPG (TD3) addresses the issue of learned Q-function dramatically overestimating Q-values in DDPG (Fujimoto et al., 2018). At the core of the improved performance of Twin Delayed DDPG is techniques including clipping double-Q learning, delayed policy updates, and target policy smoothing. To observe the actual behavior of policy gradient methods in our proposed model-based settings, we experiment on both DDPG and TD3 and compare the learning results.

Model Predictive Control (MPC) have proved efficient in predicting the future behavior of a system within a finite time horizon. Compared with traditional reactive control approaches, MPC can anticipate future disturbances and deal with non-linear systems without linearization. MPC improves the controller performance by incorporating future reference information into the control problem. Nagabandi et al. (2018a) utilized the MPC control scheme to achieve excellent sample complexity and compensate for the errors in the model. Zhang et al. (2015) combined MPC with reinforcement learning in the framework of guided policy search, using MPC to generate data at training time under full state observation in the training environment. Wang and Ba (2019) applied policy networks to generate proposals for MPC in high dimensional locomotion control problems with unknown dynamics. In order to generate meaningful virtual experiences for our model-based method, we are interested applying MPC to our setting.

3 Methods

In this section, we first briefly review the model-based reinforcement learning methods which in principle can learn more efficiently comparing to the model-free methods. Then, we describe several model-free reinforcement learning methods. Finally, we propose an algorithm that combines model-based initialization approach with model-free reinforcement learning methods.

3.1 Model-based reinforcement learning

Learning dynamics model. Neural Networks and Gaussian Processes are widely used in model dynamical functions for model-based RL. Deep neural network dynamic model performs better with complex data, while Gaussian Process is more sampling efficient dealing with simple data. We can change the model complexity by choosing between these two approaches to learn the dynamic model. The dynamics model characterizes the transitions from the current state s_t to the next state s_{t+1}. We define our dynamics model as $\hat{f}_\theta(s_t, a_t)$, and parameterize the model as a deep neural network. The parameter vector θ denotes the weights of the network. The model takes the current state s_t and action a_t as inputs and outputs an estimate of the next state $\hat{s}_{t+1}$. After data processing, we train the model $\hat{f}_\theta(s_t, a_t)$ by minimizing the error:

$$\mathcal{E}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}} \frac{1}{2} \left\| \mathbf{s}_{t+1} - \hat{f}_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right) \right\|^2 \tag{1}$$

The error provides an estimation of the performance for the learned dynamics model. After using the learned dynamics model to predict future states, we can formulate a model-based controller, which uses an MPC controller as the actor.

Model predictive control. Model predictive control (MPC) is an advanced method that has a predictive ability to take control actions accordingly. We utilize this method for the main advantage that it allows the current time slot to be optimized, meanwhile, it also keeps future time slots in the account. Based on iterative and finite-horizon optimization, MPC executes action a_t and obtains updated state s_{t+1}. Here, we use random sampling to generate candidate action sequences. At the next time step, it recalculates the optimal action sequence. MPC controller is designed for action selection during the iterative procedure to retrain the model. The reward is defined as $r(s_t, a_t)$ and $c(s_t, a_t)$ is the cost function, which can be defined by reward or quadratic function. The goal at each time step t is to take the action a_t to minimize total the cost function by solving the following optimization problem:

$$\left(\mathbf{a}_t, \ldots, \mathbf{a}_{t+H-1}\right) = \arg \max_{\mathbf{a}_t, \ldots, \mathbf{a}_{t+H-1}} \sum_{t'=t}^{t+H-1} c\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right) \tag{2}$$

In this way, the combination of the predictive dynamics model with a controller is beneficial. We conduct the model-based MPC controller as the predictive model to achieve excellent virtual experiences, which are computationally tractable. Finally, the virtual experience used by the robot arm is generated from the dynamics model, which can encode a more efficient performance on arbitrarily positional grasp tasks.

3.2 Model-free reinforcement learning

We consider a discounted, infinite-horizon, goal-conditioned Markov decision process, with states $s \in \mathcal{S}$ and goals $s \in \mathcal{G}$, action $a \in \mathcal{A}$, reward function $r(s, a, g)$, dynamics $p(s'|s, a)$ and discount factor γ. The goal of a goal-conditioned task is to maximize the expected discounted accumulated reward

$$J(\pi) = \mathbb{E}_{g \sim \rho_g, \tau \sim d^\pi(.|g)}[\sum_t \gamma^t r(s_t, a_t, g)] \tag{3}$$

under the distribution

$$d^\pi(\tau|g) = \rho_0(s_0) \prod_t \pi(a_t|s_t, g) p(s_{t+1}|s_t, at) \tag{4}$$

induced by the policy π and the initial state and goal distribution. The policy $\pi(.|s, g)$ generates continuous actions a conditioned on state s and goal g. We have goal-conditioned action-value function $Q^{\pi}(s, g, a) = \mathbb{E}_{s_0=s, a_0=a, \tau\sim d^{\pi}(.|g)}[\sum_t \gamma^t r(s_t, a_t, g)]$ and $V^{\pi}(s, g) = \mathbb{E}_{a\sim\pi}(.|s, g)Q^{\pi}(s, g, a)$. In this work we assume states and goals to co-exist in the same space, i.e. $\mathcal{S} = \mathcal{G}$.

Deep Deterministic Policy Gradients. Deep Deterministic Policy Gradients (DDPG) is an actor-critic algorithm, which concurrently learns a Q-function(critic) and a policy(actor). It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to update the policy. Check Algorithm 1 for details.

Algorithm 1 DDPG

```
Initialize empty replay buffer $D$ and target parameters $\theta_{target} \leftarrow \theta$, $\phi_{target} \leftarrow \phi$,
Observe state s and select action a = clip($\mu_\theta$(s) + $\epsilon$, $a_{Low}, a_{High}$), where $\epsilon \sim \mathcal{N}$
Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
Store $(s, a, r, s', d)$ in replay buffer D
for however many updates do
    Randomly sample a batch of transitions, B = $(s, a, r, s', d)$ from D
    Compute targets $y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{target}}(s', \mu_{\theta_{target}}(s'))$
    Update Q-function by one step of gradient descent using
        $$\nabla \frac{1}{|B|} \sum_{((s,a,r,s',d)\in B} (Q_\phi(s, a) - y(r, s', d))^2$$
    Update policy by one step of gradient ascent using $\nabla \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s))$
    Update target networks with $\phi_{target} \leftarrow \rho\phi_{target} + (1 - \rho)\phi$ and $\theta_{target} \leftarrow \rho\theta_{target} + (1 - \rho)\theta$
```

Twin Delayed DDPG. Twin Delayed DDPG (TD3) is an improved version of DDPG, it uses clipped double-Q learning, updating policy less frequently and adding noise to target action. The most important feature is clipped double-Q learning, which helps to avoid Q-values overestimation. The target value is designed as:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{target,i}}(s', a'(s')) \tag{5}$$

Universal Value Function Approximators. Universal Value Function Approximators (UVFA) Schaul et al. (2015) gives a value function approximators $V(s, g)$, that generalises not just over states s but also over goals g. Here, $s \in S$ and $g \in \mathcal{G}$. For continuous task, $\mathcal{F} : \mathcal{S} \times \mathcal{G} \to \mathbb{R}$, state and goal are concatenated together as a joint input. Therefore, we can write our value function as $V(s||g)$ and action-value function as $Q(s||g, a)$.

Hindsight Experience Replay. Hindsight experience replay (HER) Andrychowicz et al. (2017) is used to improve the robustness and sampling efficiency of goal-reaching policies, especially for sparse reward task. The basic idea is relabeling a batch of transitions with other goals given by a strategy $\mathcal{S}$. The original paper describes several goal selection strategies. We chose future strategy, which means goals selected are come from the same episode as the transition being replayed and were observed after it.

3.3 Model-based with model-free reinforcement learning

Inspired by Nagabandi et al. (2018b), we propose and implement a new algorithm called model-based MPC combined with model-Free RL. The basic idea of this algorithm is both the successful experience and virtual experience generated from learned dynamics model can speed up the DDPG training process. The details of the algorithm are as follows. In Algorithm 2, we combine the model-based algorithm MPC with the model-free algorithm DDPG. In line 5 - 7, MPC is used as an actor network to collect useful experiences. Line 9 - 11 describe how the DDPG actor networks is updated by real experiences. Line 12 updates the DDPG actor by real experience. In the end, line 13 - 14 update the DDPG actor networks by virtual experiences.

Algorithm 2 Model-based(MPC) RL with DDPG

```
 1: Dataset $D$, Transition model $T_\phi$, actor $\mu_\theta$, critic $Q_w$
 2: while not done do
 3:     Update $T_\phi$ by $D$;
 4:     for t =1 to T  do
 5:         Given $s_t$ and $g$, use $T_\phi$ estimate optimal control sequence $A_t^{(H)}$
 6:         execute $a_t$ from the first control of $A_t^{(H)}$
 7:         Store transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in D
 8:     if need update policy then
 9:         Randomly sampling minibatch $B$ from $D$
10:         for $(s_t||g, a_t, r_t, s_{t+1}||g)$ in $B$ do
11:             update $\mu_\theta$,$Q_w$ by $(s_t||g, a_t, r_t, s_{t+1}||g)$
12:             if $T_\phi$ accurate then
13:                 for $i \leftarrow 1$ to $P$ do
14:                     update $\mu_\theta$,$Q_w$ by $(s_t||g, a_t, r_t, \hat{s}_{t+1}||g)$
15:     end
```

To improve the performance of our model-based controller, based on our current algorithm, we also propose an advanced new algorithm called model-based MMPI with model-free RL adding Hindsight Experience Replay(HER). This algorithm improves the performance by enhancing the robustness to inaccuracies in the dynamics model. MPPI is a more flexible MPC method. Based on a generalized importance sampling scheme, the MPPI controller has attractive features. It is a derivative-free optimization method, and on the basis of the importance sampling of trajectories, the control sequence is iteratively updated to obtain the optimal solution. There are several advantages of the new algorithm. First, compared to random sampling in MPC, MPPI is more efficient for sampling based on the reference trajectory. Second, by introducing the DDPG actor, these two actors can benefit each other. An MPPI actor can generate a successful experience for a DDPG actor meanwhile well-trained DDPG actor can generate an efficient initial reference trajectory for MPPI. Third, adding HER buffer can make the algorithm more robust for long-term sparse reward training. The details of this algorithm are as follows.
In Algorithm 3, we combine the model-based algorithm MPPI with the model-free algorithm HER. In line 4, we select the initial reference control sequence by DDPG policy. In line 6, we add the noise to generate K candidates control sequence and similar to the MPC algorithm to solve the optimization problem and find the optimal control sequence in line 7. We use MPPI as the actor to acquire useful experiences. After we execute the first action in the selected control sequence, we need to add the action at the end of the selected control sequence to keep the same rollout horizon

and this action also select by DDPG policy in line 10. We introduce the HER in line 11 - 16 to sample a set of additional goals for replaying. The rest of the algorithm describes how we use real experiences to update DDPG actor network using real experiences and virtual experiences, which is the same as in Algorithm 2.

Algorithm 3 Model-based(MPPI) RL with DDPG+HER

```
1: Dataset $D$, Transition model $T_\phi$, actor $\mu_\theta$, critic $Q_w$
2: while not done do
3:     Update $T_\phi$ by $D$;
4:     Given $s_t$ and $g$,generate Initial control sequence $A_t^{(H)}$ by $\mu_\theta$
5:     for t =1 to T  do
6:         Adding noise to generate multiple control sequence $A_{t,i}^{(H)}$
7:         Use $T_\phi$ estimate optimal control sequence $A_{t,*}^{(H)}$
8:         execute $a_t$ from the first control of $A_{t,*}^{(H)}$
9:         Store transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in D
10:        select $a_t^{(H)}$ by $\mu_\theta$ adding to the last position of $A_{t,*}^{(H)}$
11:    episode $E = (s_1, s_2, .....s_T)$
12:    for t =1 to T  do
13:        Sample a set of additional goals for replay $G := \mathcal{S}(E)$
14:        for $g' \in G$  do
15:            $r' = r(s_t, a_t, g')$
16:            Store transition $(s_t||g', a_t, r'_t, s_{t+1}||g')$ in D
17:    if need update policy then
18:        Randomly sampling mini-batch $B$ from $D$
19:        for $(s_t, a_t, r_t, s_{t+1})$ in $B$ do
20:            update $\mu_\theta$,$Q_w$ by $(s_t, a_t, r_t, s_{t+1})$
21:            if $T_\phi$ accurate then
22:                for $i \leftarrow 1$ to $P$ do
23:                    update $\mu_\theta$,$Q_w$ by $(s_t, a_t, r_t, \hat{s}_{t+1})$
24:    end
```

4 Experiments and Results

4.1 Environment platform

Based on OpenAI gym, the environment we used for model free experiment is FetchReach-v1, which uses the Fetch research platform as the Fig. 1 shows. In this environment, the goal position is randomly chosen in a 3D space. The goal is to move the end effector of the fetch to the desired goal position. By default, reward is sparse. Only reaching goal state can return a positive reward (in this case 1), otherwise, reward is -1. The observation space is of 10 dimensions, including Cartesian position of gripper (3D), the quantity to measure the opening of gripper (2D), the velocity of gripper moving (3D), and the velocity of gripper opening/closing (2D). The action space is of 4 dimensions, including the offset in Cartesian space from the current end effector position (3D), and the state of the parallel gripper in joint space (1D). The time-steps limit are set to be 100. The environment will reset if goal is not reached within this time range.

4.2 Model free RL results

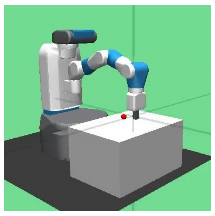
Figure 1: FetchReach-v1

In this section, we compared DDPG and TD3 performance with different environment settings and training techniques. Results shows in Fig. 2. We first fixed the goal position to be on the table, right underneath the initial position of the gripper. We then made another experiment where goal positions change randomly but reward is sense, defined as $-distance(state, goal)$. In these experiments, both DDPG and TD3 converges within 200 episodes.
We then recovered the environment to its defult settings, where goals are randomly each episode and rewards are sparse. We tested DDPG and TD3 with and without hindsight relabeling.The result shows, in a hard-exploration task like FetchReach, it is quite challenging for the agent to reach goal state within allowed 100 time-steps. Even if in one episode, it reaches a goal and get a positive reward, this experience is also hard to be taken advantage of. That's why reward almost always stays around -100. We also tested DDPG and TD3 using hindsight experience Replay Andrychowicz et al. (2017) (HER) to improve learning from sparse rewards. We used online sampling approach, which means relabeled data is not stored in the replay buffer. The result shows hindsight experience replay improve both DDPG and TD3 a great deal, with a steady reward increment. TD3 is also shown to converge much faster than DDPG.

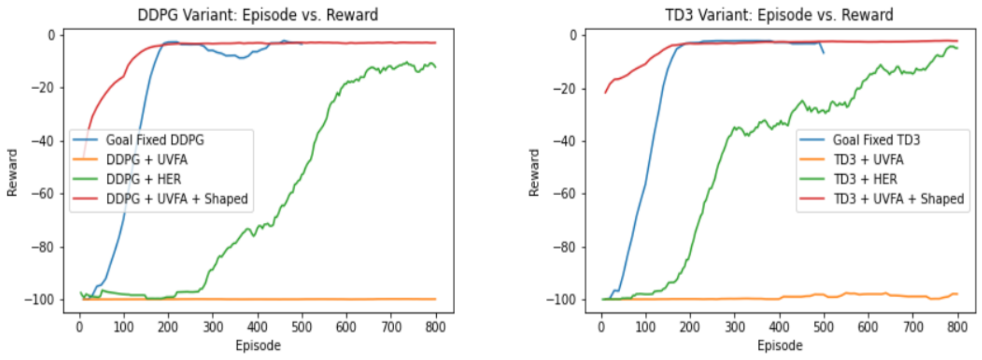

Figure 2: (a)Training result of DDPG(left) (b)Training result of TD3(right)

4.3 Model-based with model-free RL results

In this section, we will compare MB-MF learning results with Model-free learning results based on TD3. The designed experiments can be divided into three categories. First is the different task setting, in order to train robots to reach arbitrary goal points, we will set the training task as reach-break, reach-stop, and reach-new. We compare the learning performance under these three different settings. The second is experience setting. The experience collected by robots can be also mainly divided into two parts. One experience is from an MPC actor, another experience is from a TD3 actor. Thus, we will compare the learning performance by using different experiences. In the third part, we will compare the performance of the MB-MF model for both dense and sparse reward settings. Finally, we will show the results of virtual training. Here we use the advanced version of DPPG, TD3 as the model-free actor for training efficiency. Fig. 3 is model-free training results of UVFA TD3 corresponding to the dense reward and sparse reward.

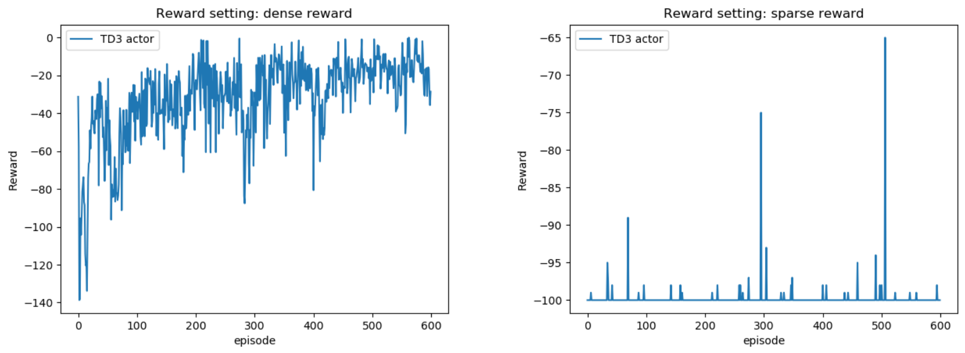

Figure 3: (a)Training result of TD3 with dense reward (b)Training result of TD3 with sparse reward

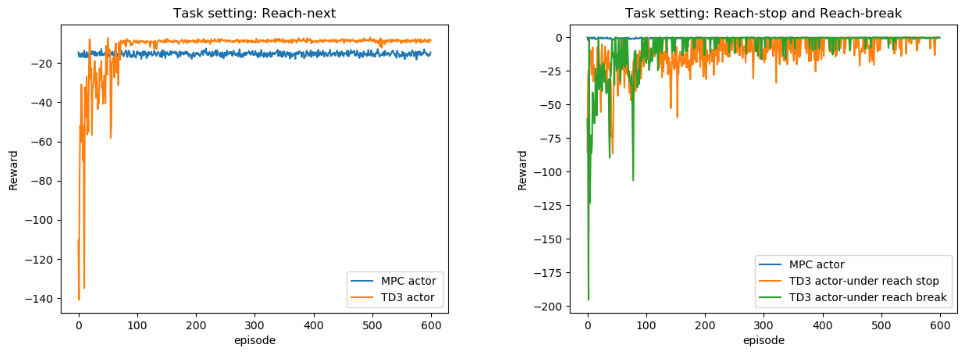

Figure 4: (a) Task setting with Reach-next (b) Task setting with Reach-stop and Reach-break

4.3.1 Task setting comparison

The training process for MB-MF model has two parts. One is the episode one which we use the MPC as the actor and another is the episode two which we choose the TD3 as the actor. For the MPC actor, as long as we have correct dynamics model and reasonable cost function. It can solve goal-reach problem efficiently. Once the robot reach the goal point, there are three options for episode one. First is called reach-break, which means once the MPC actor reach the goal point, we will break this episode and move to next episode. The second is reach-stop that means robot will keep end effector at same point when it reach the goal point. The last is called reach-next, which means we will give the new target points when the robot reach the goal points until the max time step of episode one. Based on three different task settings, the learning results are shown in Fig .4. If the task setting is reach-next as shown in Fig .4 (a), the general trend of obtaining rewards based on TD3 actor is stable after 100 episodes and it provides higher performance than MPC actor. By contrast, the general trends under the task settings of reach-stop and reach-break are not stable, fluctuating ups and downs in obtaining rewards. Besides, these results in Fig .4 (b) almost reach out to the results based on MPC actor.

4.3.2 Experience setting comparison

We conduct an experience setting comparison in Fig. 5, revolving around TD3 actor updating with only MPC experience in Fig. 5 (a) and with only TD3 experience in Fig. 5 (b). To easily compare the results, we also illustrate the line obtained by TD3 actor updating with MPC and TD3 experience in both Fig. 5 (a) and Fig. 5 (b). As we can see from the two subfigures, the combination of MPC and TD3 experience performs better than only MPC experience or only TD3 experience, showing more stable trends of obtaining rewards. Moreover, the general increasing trend of TD3 actor updating with TD3 experience is higher than that with MPC experience. The stability of TD3 actor updating with MPC experience is still expected to be improved.

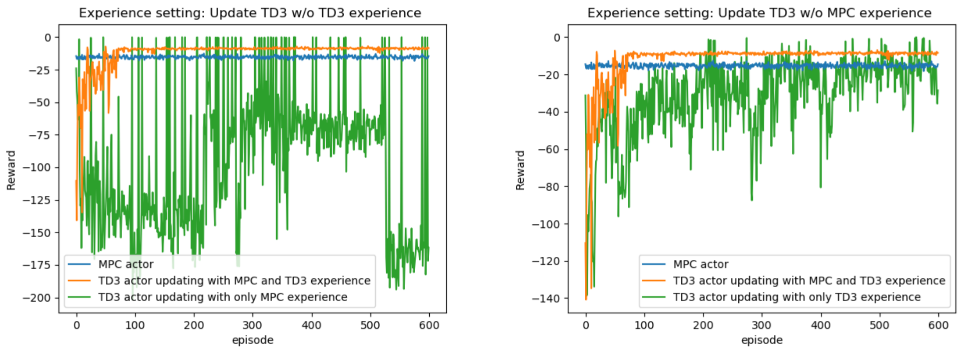

Figure 5: (a) Update TD3 w/o TD3 experience (b) Update TD3 w/o MPC experience

4.3.3 Reward setting comparison

Fig. 6 illustrates a reward setting comparison between dense reward condition and sparse reward condition. In general, the training with the setting of dense reward from Fig. 6 (a) is better than that of sparse reward from Fig. 6 (b). More specifically, we conclude that TD3 actor combined with MPC experience performs better than that without MPC experience. This conclusion can be verified from results under both dense and sparse reward conditions. Although there exist several special circumstances in Fig. 6 (a) that the rewards obtained by TD3 actor without MPC exceed rewards obtained with MPC, the erratic fluctuations are in consequence of unstable rewards. Hence, TD3 actor with MPC experience can provide more stable rewards under dense reward condition.

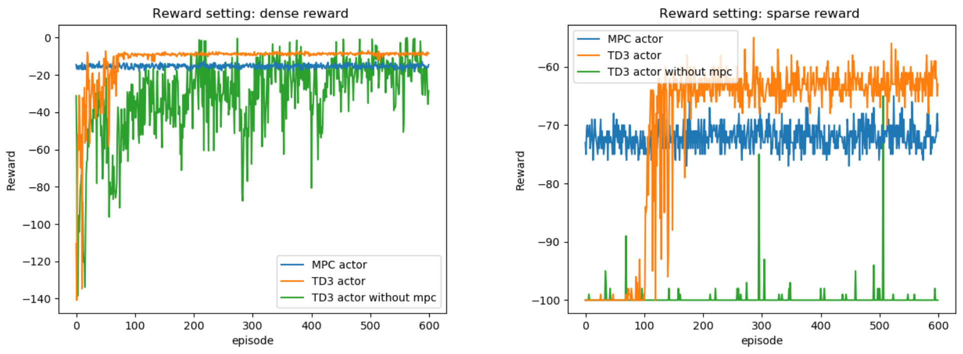

Figure 6: (a) Reward setting with dense reward (b) Reward setting with sparse reward

4.3.4 Virtual training

The most promising result for this project is that combined with MPC results, even though we only use the virtual experience to update the TD3, we still can get a well-train TD3 actor to reach an arbitrary goal position. Virtual experience means TD3 will only gain the experience from the learned environment model instead of the real environment. In other words, if in the real world, we don't need a robot to execute actions generate by the TD3 policy which might generate unreasonably and risky actions without well-trained, we can execute actions selected by the MPC

controller, collecting online data to update the real-world model and execute TD3 action in the simulation model to gain virtual experience.
As Fig. 7 shows, for both dense and sparse reward conditions, by combining the MPC experience with the TD3 virtual experience, we can get a well-trained TD3 agent. Here we compare virtual TD3 with real TD3 results. From Fig. 7 (a), we can see the reward performance of TD3 trained by virtual training has even better than TD3 trained by real experience. One possible reason for that is the virtual experience has better consistency with the MPC experience which is used to update the environment model and both TD3 agent reward performance are better than the MPC controller while the reachability of three agents are all 100%. Fig. 7 (b) is the sparse cases of virtual training, the training with sparse reward is much hard than dense reward by UVFA algorithm. However, here even for sparse reward, without Hindsight Experience Replay, we still can get the well-trained TD3 agent with real experience, and even with virtual experience, the agent reachability is also above 90%. It is trivial to compare the dense reward performance with sparse reward performance due to different metrics. For sparse conditions, once the reward is larger than the negative step length (here is -100), we can consider the robot achieving the goal position. Therefore, although the TD3 agent trained by virtual experience doesn't have a similar performance as the one trained by real experience, the virtual TD3 agent still can guarantee a high success rate of reaching arbitrary goals after 300 episodes, showing the advantage of our algorithm

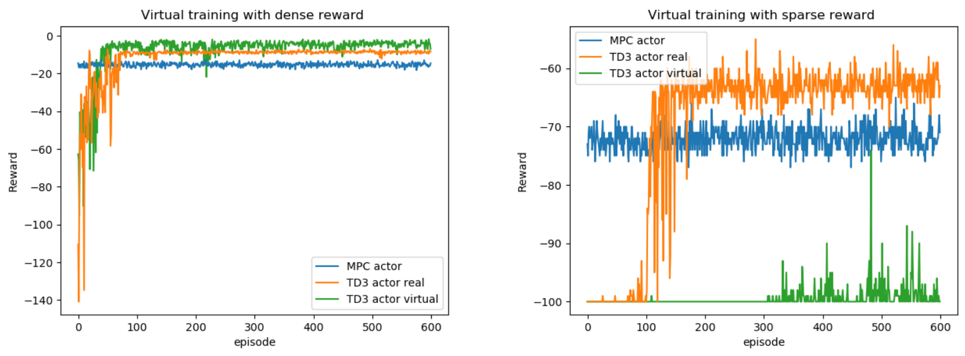

Figure 7: (a) Virtual training with dense reward (b) Virtual training with sparse reward

5 Conclusions

In this project, we train a dynamic model using real experiences and use the trained model to generate virtual experience. After acquiring these virtual and real experiences, the policy is updated using model-free approaches. We show that using our approach, the arbitrarily positional grasping problem can be solved more efficiently, compared to the existing algorithms. Furthermore, based on our current algorithm, we proposed an advanced algorithm combining MPPI with DDPG and HER in Algorithm 3. Due to the limited time, we didn't implement this idea while from the results we have, we do believe the model-based RL with a model-free algorithm can be more efficient and can apply real-time online learning in the future. In addition, there are many other interesting approaches that can be combined with and used to improve our current algorithm. Charlesworth and Montana (2020) proposed a model-based method to solve the multi-goal problem with sparse rewards. A PlanGAN, which is a variation of the Generative Adversarial Networks (GANs), is trained and used to generate plausible future trajectories. This method provides a significant increase in sampling efficiency. We think it can be used in our approach as a substitute for the Hindsight Experience Replay and could further improve the sampling efficiency.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2017. URL https://arxiv.org/abs/1707.01495.

Henry Charlesworth and Giovanni Montana. Plangan: Model-based planning with sparse rewards and multiple goals. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6101903146e4bbf4999c449d78441606-Abstract.html.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018. URL https://arxiv.org/abs/1802.09477.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *CoRR*, abs/1906.08253, 2019. URL http://arxiv.org/abs/1906.08253.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=S1xCPJHtDB.

Vikash Kumar, Emanuel Todorov, and Sergey Levine. Optimal control with learned local models: Application to dexterous manipulation. In Danica Kragic, Antonio Bicchi, and Alessandro De Luca, editors, *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, pages 378–383. IEEE, 2016. doi: 10.1109/ICRA.2016.7487156. URL https://doi.org/10.1109/ICRA.2016.7487156.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=SJJinbWRZ.

Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 1071–1079, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/6766aa2750c19aad2fa1b32f36ed4aee-Abstract.html.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 7559–7566. IEEE, 2018a. doi: 10.1109/ICRA.2018.8463189. URL https://doi.org/10.1109/ICRA.2018.8463189.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018b.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

Daniel Seita. Model-Based Reinforcement Learning: Theory and Practice. http://bair.berkeley.edu/blog/2019/12/12/mbpo, 2019. [Online; accessed 03-March-2022].

Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks, 2019. URL https://arxiv.org/abs/1906.08649.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *CoRR*, abs/1907.02057, 2019. URL http://arxiv.org/abs/1907.02057.

Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling, 2015. URL https://arxiv.org/abs/1509.01149.

Grady Williams, Andrew Aldrich, and Evangelos A. Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40 (2):344–357, 2017. doi: 10.2514/1.G001921. URL https://doi.org/10.2514/1.G001921.

Shangtong Zhang, Wendelin Boehmer, and Shimon Whiteson. Deep residual reinforcement learning. In Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith, editors, *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pages 1611–1619. International Foundation for Autonomous Agents and Multiagent Systems, 2020. URL https://dl.acm.org/doi/abs/10.5555/3398761.3398946.

Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search, 2015. URL https://arxiv.org/abs/1509.06791.

Lixin Zou, Long Xia, Pan Du, Zhuo Zhang, Ting Bai, Weidong Liu, Jian-Yun Nie, and Dawei Yin. Pseudo dyna-q: A reinforcement learning framework for interactive recommendation. In James Caverlee, Xia (Ben) Hu, Mounia Lalmas, and Wei Wang, editors, *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pages 816–824. ACM, 2020. doi: 10.1145/3336191.3371801. URL https://doi.org/10.1145/3336191.3371801.